

This Page Is Inserted by IFW Operations  
and is not a part of the Official Record

## **BEST AVAILABLE IMAGES**

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

**IMAGES ARE BEST AVAILABLE COPY.**

**As rescanning documents *will not* correct images,  
please do not report the images to the  
Image Problems Mailbox.**



```
certItem = 0;  
delete () alicodes;  
aLICodes = 0;  
aICodes = null;  
delete () locations;  
nLocations = 0;  
locations = null;  
targetPages.Remove();  
targetSites.Remove();  
siteCategories.Remove();  
interests.Remove();  
addDescription.Empty();  
fillName.Empty();  
jumpTo.Empty();  
}  
endit
```

HIGHLY  
CONFIDENTIAL  
DC 069521

```

000L Ad.Book( DMOD advertiserID )
{
    char buf(1024);
    char strtime[ 30 ];
    if ( !advertiserID )
    {
        ASSERT( 0 );
        return( FALSE );
    }

    // If this is a banner ad, set max_impressions = 1
    // if (type == BANNER)
    {
        max_impressions = 1;
    }

    strcpy( buf, "insert placements(jumpTo,max_impressions,type,po,browser,domainType,lep,freq=
    "image,series,advertiser,flags,hours_of_day,days_of_week,employees,sales,descr=
    "max_amount,po_number,gender,active,approved,filename" );

    if ( !startTime )
        strcpy( buf, "start_time" );

    if ( !endTime )
        strcpy( buf, "end_time" );

    addvalue( buf, "jumpTo" );
    addvalue( buf, "max_impressions" );
    addvalue( buf, "type" );
    addvalue( buf, "po" );
    addvalue( buf, "browser" );
    addvalue( buf, "domainType" );
    addvalue( buf, "lep" );
    addvalue( buf, "frequency" );
    addvalue( buf, "imageSeries" );
    addvalue( buf, "advertiserID" );
    addvalue( buf, "flags" );
    addvalue( buf, "hoursOfDay" );
    addvalue( buf, "daysOfWeek" );
    addvalue( buf, "nEmployees" );
    addvalue( buf, "salesVolume" );
    addvalue( buf, "adDescription" );
    addvalue( buf, "maxAmount" );
    addvalue( buf, "poNumber" );
    addvalue( buf, "gender" );
    addvalue( buf, "active" );
    addvalue( buf, "approved" );
    addvalue( buf, "filename, FALSE );

    if ( !errorCode )
    {
        strcpy( buf, " " );
        strcpy( buf, " " );
        addvalue( buf, "strtime, FALSE );
    }

    if ( !endTime )
    {
        strcpy( buf, " " );
        strcpy( buf, " " );
        addvalue( buf, "strtime, FALSE );
    }

    if ( !endTime )
    {
        strcpy( buf, " " );
        if ( !ifmain_exact buf )
        {
            ASSERT( 0 );
            return( FALSE );
        }
    }
}

```

DC 069518

HIGHLY  
CONFIDENTIAL

```

// Get the ID of the newly added ad
int adid = 0;

{
    Cursor c;
    c.bind( SQL_C_LONG, adid, 4 );
    strcpy( buf, "select max(id) from placements" );
    c.execute( buf );
    c.fetchFirst();
    ifmain.Commit();
}

if ( !adid )
{
    ASSERT( 0 );
    return( FALSE );
}

return( addPlacementTable( adid ) );

000L Ad.Update()
{
    // To update an ad, we delete the existing ad
    // and re-book it.
    if ( !remove( FALSE ) )
    {
        // Determine if the ad is targeted
        double dPeradCost = CalculateCostPeriod();
        if ( !dPeradCost == BASE_AD_COST )
        {
            flags = Ad.Targeted;
        }
        else
        {
            flags = Ad.Targeted;
        }
    }

    char buf(1024);
    char strtime[ 30 ];

    strcpy( buf, "update placements set " );

    // Don't update max_impressions if this is a banner ad. REP. ETC.
    // credits the placement so we don't want to overwrite the
    // banner credits
    if ( type != BANNER )
    {
        strcpy( buf, "max_impressions=" );
        addvalue( buf, "max_impressions" );

        strcpy( buf, "jumpTo=" );
        addvalue( buf, "jumpTo" );
        strcpy( buf, "type=" );
        addvalue( buf, "type" );
        strcpy( buf, "po=" );
        addvalue( buf, "po" );
        strcpy( buf, "browser=" );
        addvalue( buf, "browser" );
        strcpy( buf, "domainType=" );
        addvalue( buf, "domainType" );
        strcpy( buf, "lep=" );
        addvalue( buf, "lep" );
        strcpy( buf, "frequency=" );
        addvalue( buf, "frequency" );
        strcpy( buf, "imageSeries=" );
        addvalue( buf, "imageSeries" );
        strcpy( buf, "hours_of_day=" );
        addvalue( buf, "hoursOfDay" );
        strcpy( buf, "days_of_week=" );
        addvalue( buf, "daysOfWeek" );
        strcpy( buf, "employees=" );
        addvalue( buf, "nEmployees" );
        strcpy( buf, "sales=" );
        addvalue( buf, "salesVolume" );
        strcpy( buf, "adDescription=" );
        addvalue( buf, "adDescription" );
        strcpy( buf, "max_amount=" );
        addvalue( buf, "maxAmount" );
        strcpy( buf, "po_number=" );
        addvalue( buf, "poNumber" );
        strcpy( buf, "gender=" );
        addvalue( buf, "gender" );
        strcpy( buf, "active=" );
        addvalue( buf, "active" );
        strcpy( buf, "approved=" );
        addvalue( buf, "approved" );
        strcpy( buf, "filename=" );
        addvalue( buf, "filename" );

        if ( !errorCode )
        {
            if ( !errorCode )
            {
                strcpy( buf, "start_time=" );
                addvalue( buf, "start_time" );
            }
        }
    }
}

```

```

        {
            setctime( atime, 9, "mm/dd/yy", getctime( atertime ) );
            addvalue( buf, atime );
        }
        else
        {
            setctime( buf, "null");
        }
        setctime( buf, "end_time");
        if (endtime)
            setctime( atime, 9, "mm/dd/yy", getctime( endtime ) );
            addvalue( buf, atime, FALSE );
        }
        else
        {
            setctime( buf, "null");
        }
        setctime( buf, "where id=" );
        addvalue( buf, id, FALSE );
        if ( !ismain.exec( buf ) )
        {
            ASSERT( 0 );
            return FALSE;
        }
        return( AddPlacementTable( id ) );
    }
    return FALSE;
}

bool Ad::AddPlacementTable( DWORD adid )
{
    char buf(1024);
    BOOL brc = TRUE;
    while (TRUE)
    {
        // Now save the locations to the "placement_locations" table
        // for the nloop - 0; nloop < nLocations; nloop++
        {
            strcpy( buf, "insert placement_locations(" );
            if ( !locations[nloop].country )
                setctime( buf, "country=" );
            if ( !locations[nloop].state.isEmpty() )
                setctime( buf, "state=" );
            if ( !locations[nloop].zipcode.isEmpty() )
                setctime( buf, "zipcode=" );
            if ( !locations[nloop].areacode )
                setctime( buf, "areacode=" );
            setctime( buf, "ad_id) values(" );
            if ( !locations[nloop].country )
                addvalue( buf, locations[nloop].country );
            if ( !locations[nloop].state.isEmpty() )
                addvalue( buf, locations[nloop].state );
            if ( !locations[nloop].zipcode.isEmpty() )
                addvalue( buf, locations[nloop].zipcode );
            if ( !locations[nloop].areacode )
                addvalue( buf, locations[nloop].areacode );
            addvalue( buf, adid, FALSE );
            setctime( buf, " );" );
            if ( !ismain.exec( buf ) )
            {
                ASSERT( 0 );
            }
        }
    }
}

```

DC 069519

HIGHLY  
CONFIDENTIAL

```

        brc = FALSE;
        break;
    }
    // Now save the sites to the "placement_sites" table
    // for nloop - 0; nloop < nSiteCodes; nloop++
    {
        strcpy( buf, "insert placement_sites(ad_id,sitecode) values(id," );
        adid, sitecode);
        setctime( buf, " );" );
        if ( !ismain.exec( buf ) )
        {
            ASSERT( 0 );
            brc = FALSE;
            break;
        }
    }
    // Now save the site categories to the placement_sitecat table
    // POSITION pos = siteCategories.GetStartPosition();
    DWORD dwInterestID;
    BOOL bBunk;
    while (pos)
    {
        siteCategories.GetNextAssoc( pos, dwInterestID, bBunk );
        strcpy( buf, "insert placement_sitecat(ad_id,interest_id) values(id," );
        adid, dwInterestID );
        setctime( buf, " );" );
        if ( !ismain.exec( buf ) )
        {
            ASSERT( 0 );
            brc = FALSE;
            break;
        }
    }
    // Now save the user interests to the placement_interests table
    // pos = interests.GetStartPosition();
    while (pos)
    {
        interests.GetNextAssoc( pos, dwInterestID, bBunk );
        strcpy( buf, "insert placement_interests(ad_id,interest_id) values(id," );
        adid, dwInterestID );
        setctime( buf, " );" );
        if ( !ismain.exec( buf ) )
        {
            ASSERT( 0 );
            brc = FALSE;
            break;
        }
    }
    // Now save the target sites to the placement_target_sites table
    // pos = targetSites.GetStartPosition();
    while (pos)
    {
        targetSites.GetNextAssoc( pos, dwSiteID, bBunk );
        strcpy( buf, "insert placement_target_sites(ad_id,site_id,include) values(id," );
        adid, dwSiteID, include );
        setctime( buf, " );" );
        if ( !ismain.exec( buf ) )
        {
            ASSERT( 0 );
        }
    }
}

```

```
// sitepage.cpp

#include "data.h"
#include "object.h"
#include "db.h"
#include "db.h"
#include "db.h"
#include "db.h"

void message(const char *)

SitePage::SitePage()
{
    id = 0;
    siteid = 0;
    categorized = FALSE;

    void SitePage::loadCategories()
    {
        DBCOND InteractID;
        Cursor ci;
        c.bind(SQL_C_LONG, InteractID, sizeof(InteractID));
        char sql[512] = "select Interact_id from page_categories where page_id=";
        addValue(buf, id, FALSE);
        strcat(sql, " union all select Interact_id from site_categories where site_id=");
        addValue(buf, siteid, FALSE);
        strcat(sql, " ");
        while (c.fetchNext()) {
            categories.Add(InteractID);
        }
    }

    extern BOOL defaultAdMode;

    SitePage::loadPage(Database db, const char *from, const char *requestId)
    {
        // from key format: sitekey/docname
        if (from == 0)
            return 0;

        if (strlen(from, "www.", 4) == 0)
            from = "";

        if (strlen(from) == 0)
            return 0;
        const char *q = strchr(from, '/');
        if (q == 0 || strlen(from) > 75)
            return 0;

        CString key;

        // truncate a unique number from the end of the key
        const char *lastSlash = strchr(q, '/');
        if (lastSlash != 0)
            key = CString(from, lastSlash - from);
        else
            key = from;

        if (key.GetLength() > 64)
            key = key.Left(64); // truncate to column width

        SitePage *p = new SitePage;

        Cursor ci(db);
        c.bind(SQL_C_LONG, sp-siteid, 4);
        c.bind(SQL_C_LONG, sp-siteid, 4);
        c.bind(SQL_C_LONG, sp-siteid, 4);
        c.bind(SQL_C_LONG, sp-siteid, 4);
        char sql[512] =
            "select id,site,categorized from sitepage where keyname=";
        addValue(buf, key, FALSE);
        c.execute();
        if (c.fetchNext()) {
            return p;
        }
    }
}
```

DC 069516

HIGHLY  
CONFIDENTIAL

```
// Didn't find the page. Add page if site is correct.
CString siteKey(from, q - from);
int approved = 0;
Cursor ci(db);
c.bind(SQL_C_LONG, sp-siteid, 4);
c.bind(SQL_C_LONG, sp-siteid, 4);
c.bind(SQL_C_LONG, sp-siteid, 4);
CString sql = "select id,approved from sites where keyname=";
sql += siteKey + " ";
c.execute();
if (c.fetchNext()) {
    if (approved == 0) {
        message CString("unapproved site ") + from;
    }
    else {
        p->add(db, key);
    }
}
else {
    delete p;
    p = 0;
    if (defaultAdMode)
        message CString("unknown site ") + from;
}

return p;

void SitePage::add(Database db, const char *keyname)
{
    char buf[512] = "insert sitepage(junk, keyname, site, categorized) values('";
    addValue(buf, keyname);
    addValue(buf, (int) siteid);
    addValue(buf, (int) categorized, FALSE);
    strcat(buf, "')";
    if (db.execute(buf) != 1) {
        TRACE("error adding sitekey\n");
        CString s = "sql: ";
        s += buf;
        ASSERT(FALSE);
        TRACE(s);
        message(s);
    }
}

Cursor ci(db);
id = 0;
c.bind(SQL_C_LONG, id, 4);
strcpy(buf, "select id from sitepages where keyname=");
addValue(buf, keyname, FALSE);
c.execute();
if (c.fetchNext()) {
}
```

```
// ad.cpp
//
#include "cdat.h"
#include "cstrtree.h"
#include "cstream.h"
#include "winsock.h"
#include "object.h"
#include "/d:/toolkit/lib/util.h"
#include "/d:/toolkit/db.h"
#include "/d:/toolkit/dbutil.h"
#include "/d:/deriver/qjderiver.h"
#include "/d:/deriver/qjdr.h"
#include "remembered.h"

const CString gIlsFootCdr = "C:\\len\\ad\\";

if (!defined(DRIVER))
int main() { return ads.GetSize(); }

endif

extern Database InfoMain;

// .....
// Ad

Ad::Ad()
{ delete[] locations;
delete[] siccodes;

Ad::Ad(const Ada ad) :
started(ad.started),
id(ad.id), fileName(ad.fileName), jumpTo(ad.jumpTo),
type(ad.type), os(ad.os), browser(ad.browser),
domainType(ad.domainType), lpt(ad.lpt),
mailImpression(ad.mailImpression), nshom(ad.nshom),
nicrAtLine(ad.nicrAtLine), nsICCode(ad.nsICCode),
frequency(ad.frequency), imageSort(ad.imageSort),
serName(ad.serName), startTime(ad.startTime),
startTime(ad.startTime),
startDay(ad.startDay), dayOfWeek(ad.dayOfWeek),
hourOfDay(ad.hourOfDay), salaryVolume(ad.salaryVolume),
nEmploYes(ad.nEmploYes), salaryVolume(ad.salaryVolume),
gender(ad.gender), address(ad.address),
salaryVolume(ad.salaryVolume), salaryVolume(ad.salaryVolume),
active(ad.active), includePages(ad.includePages), approved(ad.approved),
nJumped(ad.nJumped)

strcpy(fileName);
strcpy(sicCodes[jumpTo]);

locations = 0;
if (!locations) {
locations = new Region(locations);
for(int i = 0; i < nLocations; i++) {
locations[i] = ad.locations[i];
}
}

siccodes = 0;
if (!siccodes) {
siccodes = new siccodes(sicCodes);
for(int i = 0; i < nsicCodes; i++) {
siccodes[i] = ad.sicCodes[i];
}
}

void Ad::calcSII()
{ if (!mailImpressions == 0) {
return;
}
```

```

time_t t;
DWORD totalSpan = endTime - startTime;
if (totalSpan == 0)
    totalSpan = 1;
DWORD span = (time_t) (totalSpan / (span == 0 ? span : 1));

sl =
    (DWORD) ((double) nShown /
    (double) span / totalSpan) /
    maxImpressions * 1000);

}

void Ad::AdShown()
{
    nShown++;
    // if (nShown % 5 == 0) {
    //     // update SI
    //     calcSI();
    // }
}

Ad::Ad()
{
    dayOfWeek = 0x7f;
    started = FALSE;
    flags = Production | SpreadEvenly;
    sl = 1100;
    nClickCodes = 0;
    nClickCodes = 0;
    frequency = 0;
    imageSeries = FALSE;
    id = 0;
    maxImpressions = 0;
    nShown = 0;
    nViews = 0;
    type = Normal;
    nClickCodes = 0;
    nClickCodes = 0;
    gender = 0;
    payment = 0;
    active = 0;
    approved = 0;
    includePages = 0;
    includeSites = 0;
    startTime = 0;
    endTime = 0;
    os = DefaultMask;
    browser = DefaultMask;
    domainType = DefaultMask;
    lnp = DefaultMask;
    hoursOfDay = 0x7f;
    nEmployees = DefaultMask;
    nEmployees = DefaultMask;
    gender = DefaultMask;
    serialment = 0;
}

CString Ad::getFileName()
{
    if (imageSeries || serialment == 1)
        return fileName;
    char buf[256];
    sprintf(buf, "%d.gif", (const char *) (fileName.Left(fileName.GetLength() - 4).serialment));
    return buf;
}

CString Ad::FullName()
{
    return getFileRoot() + getFileName();
}

}

if (defined(_MSVP))

```

```

// users.cpp
#include "objects.h"
#include "d/cookie/db.h"
#include "d/cookie/lat_util.h"
#include "d/cookie/dbutil.h"

/* Implementation for hash tables
user: User::lookupUserByIP(DWORD userIP)
return u:
{
    User u;
    lookupUserByIPAddress(DWORD ip)
    {
        DWORD userIP = networkToHostIp(userIP, FALSE);
        if (userIP == 0) {
            // Try to get domain info at least. Note: if user is uniquely
            // identifiable, derive data process will create a record for the
            // user as soon as it gets a chance.
            userIP = networkToHostIp(userIP, TRUE);
        }
        if (userIP) {
            return lookupUserByIPID(userIP);
        }
        return 0;
    }
}

class UserCursor : public Cursor
{
public:
    UserCursor(Database db, User *u) : Cursor(db),
        u(u) {}

    // Just gets field that aren't derivable from request header
    void minimalBind()
    {
        bind( SQL_C_LONG, u->ipStrid, sizeof(BOOL) );
        bind( SQL_C_LONG, u->hasCookie, sizeof(BOOL) );
    }

    User *u;

    void User::lookupUserByIPInfo(Database db)
    {
        if (userIP == 0) {
            return;
        }

        Cursor c(db);
        char sql[128];
        sprintf(sql, "select email from users where id=id", userIP);
        c.bind(emailStrid);
        c.execute();
        c.fetchNext();
        db.commit();
    }

    User *u = new User;
    UserCursor c(db, u);
    c.minimalBind();
    char sql[128];
    sprintf(sql, "select ftp_cried, has_cookie from users where id=id", userIP);
    if (timedOut != 0)
        c.setTimedOut(1);
    c.execute();
}

```

DC 069514

HIGHLY  
CONFIDENTIAL

```

if (c.timedOut()) {
    "timedOut = TRUE;
    delete u; u = 0;
}
else if (c.fetchNext()) {
    u->userIP = userIP;
}
else {
    delete u;
    u = 0;
}

return u;

User *User::lookupUserByIPAddress(Database db, DWORD ip, BOOL *timedOut)
{
    User *u = new User;
    UserCursor c(db, u);
    c.minimalBind();
    c.bind( SQL_C_LONG, u->userIP, 4 );
    char sql[128];
    sprintf(sql, "select ftp_cried, has_cookie, id from users where ip=%d",
        (const char *) sqlToStr(ip) );
    if (timedOut != 0)
        c.setTimedOut(1);
    c.execute();

    if (c.timedOut()) {
        "timedOut = TRUE;
        delete u;
        u = 0;
    }
    else if (c.fetchNext()) {
        delete u;
        u = 0;
    }
    return u;
}

void User::updateIPStrid(Database db)
{
    if (tempUserObject()) {
        ASSERT(FALSE);
        return;
    }

    char buf[128];
    sprintf(buf, "update users set ftp_cried=id where id=id",
        ipStrid );
    db.execute(buf);
    db.commit();
}

void User::makePermanent(Database db)
{
    if (tempUserObject())
        return;

    ASSERT name.isEmpty() && title.isEmpty() && emailAddr.isEmpty();

    // add to DB
    char buf[1024];
    sprintf(buf, "insert users (ip, browser, dver1, dver2, or_domain_type, is_proxy, is_network_desc, ftp_cried, has_cookie) values ('%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s')",
        sqlToStr(ip), sqlToStr(browser), sqlToStr(dver1), sqlToStr(dver2), sqlToStr(or_domain_type), sqlToStr(is_proxy), sqlToStr(is_network_desc), sqlToStr(ftp_cried), sqlToStr(has_cookie));
    addValue(buf, browser);
    addValue(buf, dver1);
    addValue(buf, dver2);
    addValue(buf, or_domain_type);
    addValue(buf, is_proxy);
    addValue(buf, is_network_desc);
    addValue(buf, ftp_cried);
}

```

users.cfp

29-Dec-1993 16:52

Page 3(3)

```
addbool(buf, hasCookie, FALSE);
arecal(buf, "-");
if (db.doinser(buf) == 1) {
  Cursor c(db);
  c.bind(SQL_C_LONG, userID, 4);
  arecal(buf, "select max(id) from users where ip=");
  addIntValue(buf, ip, FALSE);
  c.exec(buf);
  c.fetchNext();
  ASSERT( userID != 0 );
}
db.commit();
```

HIGHLY  
CONFIDENTIAL  
DC 069515

```

// server.cpp
//
#include "stdafx.h"
#include "stream.h"
#include "server.h"
#include "d/coolkit/socket.h"
#include "d/coolkit/messages.h"
#include "d/coolkit/stdlib.h"
#include "d/coolkit/ADVM.h"
#include "getrequest.h"
#include "tables.h"
#include "iconat.h"
#include "defnidl_IAP"
#include "request.h"
#include "lairequest.h"
void qfPurge();
void message(iconat char *) ( )
false
#include "request.h"
#include "mgntrequest.h"
void message(iconat char *) ( )
endl;
#include "d/coolkit/crit.h"
extern CriticalSection task;

const char HTTPVer[] = "HTTP/1.0 ";
const char cContentLen[] = "Content-length: ";
const char cReqHeader[] = "<h>";
const char cErrHeader[] = "<h>";
const char cErrTrailer[] = "Content-Type: text/html\r\n";

extern int nLastErrorHeader;

ofstream errLog;

void sendError(Connection *c, const char *msg, const char *headerField)
{
    char buf[10];

    CString s = HTTPVer;
    s += msg;
    s += "\r\n";
    s += cContentLen;
    s += cErrHeader;
    s += headerField;
    int len = strlen(s);
    s += cErrTrailer;
    s += cErrHeader;
    s += "\r\n\r\n";
    s += msg;
    s += cErrTrailer;
    c->write( (const char *) s, s.GetLength());
}

bool addressOK(const sockaddr_in from)
{
    if( (from.sin_addr.s_un.s_un_b.b1 == 206 &&
        from.sin_addr.s_un.s_un_b.b2 == 4 &&
        from.sin_addr.s_un.s_un_b.b3 == 319) )
    {
        // IAP network
        return TRUE;
    }
    return FALSE;
}

void serviceRequest(Connection *c, const sockaddr_in from)
{
    // if( !addressOK(from) )
    //     return;
}

```

DC 069512  
CONFIDENTIAL  
HIGHLY

```

const BUFLEN = 32768;
char buf[BUFLEN];
buf[0] = 0;

// total n bytes read
int n = 0;
const char *p = buf;

int countDown = 0;
// Content-Length
int connectionReadError err = Connection::OK;
while (1) {
    int toRead = BUFLEN - n - 1;
    int nread = c->read(buf + n, toRead, err);
    n += nread;
    buf[n] = 0;
    if (countDown == 0)
        countDown = nread;
    if (countDown == 0)
        break;
}

if (nread == 0) {
    // error
    break;
}

const char *p;
if (ip = strstr(buf, "\n\n\r\n")) p = 0; {
    const char *c1 = strstr(buf, "Content-Length:");
    if (!c1)
        c1 = strstr(buf, "ContentLen");
    if (c1) {
        c1 += 15;
        sscanf(c1, "%d", &countDown);
        countDown = strlen(ip + 4); // decrement by what we've already got
        countDown = n - (ip + 4) - buf[1]; // decrement by what we've already got
        if (countDown > 0)
            continue;
        break;
    }
}

Verb v = UNKNOWN;
const char *r = buf;
if (strstr(buf, "GET ") == 0) {
    v = GET;
} else if (strstr(buf, "HEAD ") == 0) {
    v = HEAD;
} else if (strstr(buf, "POST ") == 0) {
    v = POST;
} else if (strstr(buf, "PUT ") == 0) {
    v = PUT;
} else if (v == UNKNOWN) {
    if (buf == 0) {
        sendError(c, "400 bad request");
        if (buf[1] != 0) {
            message("empty request, buf[1]");
        } else if (err == Connection::Timeout) {
            message("empty request, timeout");
        } else if (err == Connection::ReadErr) {
            message("empty request, readerr");
        } else {
            message("empty request, err=OK?");
        }
    }
    else {
        sendError(c, "501 Not Implemented");
    }
}

return;

```

```

        alt defined(_JAF)
        JAFRequest grfc_v_r_from;
        alt defined(_ADJVM)
        alt defined(grfc_v_r_from);
        return grfc_v_r_from;
    }

    Httprequest grfc_v_r_from;
    sendit
    gr.service();
}

classur classenur = 0;

test: nthread = 0;
int mainthreads = 1;

JMT listenerthread(LPVOID)
{
    static DWORD ed = GetTickCount();
    sendit ed;
}

while(1) {
    socketaddr_in from;
    Connection c = listener_waitForConnection(from);
    if(c) {
        {
            Crit c(fact);
            int n = nthread;
            if(n > mainthreads)
                mainthreads = n;
        }
        serviceRequest(c, from);
        delete c;
    }
    Crit c(fact);
    nthread--;
    alt defined(_JAF)
    alt defined(_JAF)
    if(nthread == 0) {
        // idle
        qpurge();
    }
}

sendit
}

return 0;
}

bool startServer()
{
    alt defined(_ADJVM)
    alt defined(_ADJVM)
    alt defined(_Error opening tables");
    return false;
}

if(!initWinsock()) {
    return false;
}

main:main();
initComerTimezoneTable();
sendit
}

if(0
{
    // TDUP!
    Connection c;
    if(c.connect("www.microsoft.com", 80)) {
        c.write("GET /sdd HTTP/1.0\r\n\r\n", 23);
        while(1) {
            char buf[256];
            int n = c.read(buf, 255);

```

DC 069513

**HIGHLY  
CONFIDENTIAL**

```

if (n) {
    buf[0] = 0;
    TRACE("%s", buf);
}
else
    break;
}
return TRUE;
}

sendit
{
    if (defined(_PORT))
        int port = _PORT;
    else
        int port = 80;

    sendit
        listener = new Listener(port);
        if (listener->ok()) {
            if (defined(_ADDR))
                errlog.open("%s/bin/errlog.txt",
                    logdir | localapp,
                    ("bufinb read"));
            ASSERT( errlog.is_open() );
            errlog << "-----ad server started\n"; errlog.flush();
            sendit
                for (int i = 0; i < listenerThreads; i++) {
                    sleep(100); // [dam] this is a test; sometimes it doesn't listen right, just a bunch
                    AtmeginThread( listenerThread, 0 );
                }
            else
                ASSERT(FALSE);
        }
    return TRUE;
}

```





```

// sqldb.cpp
#include "stdafx.h"
#include "stream.h"
#include "objects.h"
#include "d/coolkit/db.h"
#include "d/coolkit/inf_util.h"
#include "d/coolkit/dbutil.h"
#include "d/coolkit/dbpool.h"
#include "d/coolkit/critic.h"

// this ad displayed if a bad eickey is encountered
const char keyAdID = 49;

extern CriticalSection (cs);

Database lafmain;
void message(const char *);
BOOL defaultAdMode = FALSE;

static int ucToOf;
static void localOutCritic(cs)
{
    ucToOf;
}

// This is temporary, used for non-unique users.
// Eventually will be smarter about what to send to
// these users.
Ad *defaultAd = 0;

Ad *badKeyErrorAd = 0;

typedef CArray<Ad *, Ad *, ADArray>
    SQLDB AdArray;

SQLDB loadAdArray(Ad *);
DWORD advertiseID;
// if forgetting, update AdTargets to reflect
// current forgetting.
// site actions only
// active only
// include where enddate has past or where all delivered
// newest first.
// order from newest to oldest
DWORD oldest = 0;

SQLDB openSQLDB()
{
    lafmain.open();
    openDBPool();
    // if (lafmain.open())
    // return FALSE;
    // if (openDBPool())
    // return FALSE;
    // if (lafmain.open(), FALSE, FALSE,
    // "ODBCDSN=\\SQLSERVER\\PMD",
    // "FALSE", TRUE)
    // return FALSE;
    // if (loadAdArray(0, TRUE, TRUE, FALSE, FALSE))
    // return FALSE;
    return TRUE;
}

void reloadAd()
{
    BOOL ok = FALSE;
    message("waiting to reload ads...");
    AFRGetApp().m_pMainWnd->Validate();
    AFRGetApp().m_pMainWnd->UpdateWindow();
    while (1)
    {

```

DC 069508

HIGHLY  
CONFIDENTIAL

```

{
    Crit c(fact);
    if (allFree()) {
        for (int i = 0; i < AdArray.GetSize(); i++) {
            delete AdArray[i];
            AdArray.RemoveAll();
            defaultAd = 0;
            ok = loadAdArray(0, TRUE, TRUE, FALSE, FALSE);
            break;
        }
        Sleep(50);
    }
    if (ok)
        message("Ad reload completed OK");
    else
        message("Ad reload failure");
}

// note: this isn't getting called yet
void closeSQLDB()
{
    lafmain.close();
}

//
// Ad
AdArray ads;

class AdCursor : public Cursor
{
public:
    AdCursor()
    {
        bind(SQL_C_LONG, ad.id, 1);
        bind(SQL_C_LONG, ad.type, sizeof(ad.type));
        bind(SQL_C_LONG, ad.or, sizeof(ad.or));
        bind(SQL_C_LONG, ad.browser, sizeof(ad.browser));
        bind(SQL_C_LONG, ad.domainType, sizeof(ad.domainType));
        bind(SQL_C_LONG, ad.lsp, sizeof(ad.lsp));
        bind(ad.litname);
        bind(ad.jumpTo);
        bind(SQL_C_LONG, ad.frequency, sizeof(ad.frequency));
        bind(SQL_C_LONG, ad.imageSeries, sizeof(ad.imageSeries));
        bind(SQL_C_LONG, ad.maxImpressions, sizeof(ad.maxImpressions));
        bind(SQL_C_LONG, ad.nshown, sizeof(ad.nshown));
        bind(SQL_C_LONG, ad.startTime, sizeof(ad.nshown));
        bind(SQL_C_LONG, ad.endTime, sizeof(ad.nshown));
        bind(SQL_C_LONG, ad.flags, sizeof(ad.flags));
        bind(SQL_C_LONG, ad.hourOfDay, sizeof(ad.hourOfDay));
        bind(SQL_C_LONG, ad.dayOfWeek, sizeof(ad.dayOfWeek));
        bind(SQL_C_LONG, ad.dayOfMonth, sizeof(ad.dayOfMonth));
        bind(SQL_C_LONG, ad.volume, sizeof(ad.volume));
        bind(SQL_C_LONG, ad.active, sizeof(ad.active));
        bind(ad.description);
        bind(SQL_C_LONG, ad.maxAmount, sizeof(ad.maxAmount));
        bind(ad.sponsored);
        bind(SQL_C_LONG, ad.approved, sizeof(ad.approved));
        bind(SQL_C_LONG, ad.dumps, sizeof(ad.dumps));
    }
}

Ad ad;

// ... TODO!!! This function is not thread-safe.
void reloadAd()
{
    for (int i = 0; i < AdArray.GetSize(); i++) {
        Ad ad = AdArray[i];
        ad.Calc();
    }
}

```



```
void Request::service()
{
    const char *p = strchr(request, ' ');
    if (p)
        filename = &strchr(request, p - request);
    else
        filename = request;

    {
        const char *p = filename;
        if (*p == '/')
            p++;
        if (*p == 0)
            // send default
            // sendfile("h:\\my documents\\internet address (under\\lafmain.htm");
            if (!defined(IAP))
                sendfile("c:\\laf\\html\\lafmain.htm");
            return;
        sendit
    }
    else {
        if (strcmp(p, "\\") == 0 || strcmp(p, ".") == 0) {
            if (strcmp(p, "/") != 0) {
                CString t = "c:\\lan\\";
                t += p;
                sendfile(t);
                return;
            }
            else {
                if (!defined(IAP))
                    CString t = "c:\\laf\\html\\";
                else if (!defined(MANAGE))
                    CString t = "c:\\lan\\manage\\";
                else
                    ASSERT(FALSE);
                CString t = "jehid";
                //CString t = "h:\\my documents\\ad federation\\";
                sendit
            }
            {
                t += p;
                sendfile(t);
                return;
            }
        }
        senderror("404 Not Found");
    }
}

void Request::sendInternalError()
{
    senderror("500 Internal Server Error");
}
```

DC 069506

HIGHLY  
CONFIDENTIAL

```
// remembered.cpp
//
#include "cdat.h"
#include "objects.h"
#include "remembered.h"
#include "d/coolkit/hash.h"
#include "d/coolkit/crit.h"

const SZ = 107333;

// this is a test
static int cri
#define INCRIT ( ASSERT(cri==0), cri++ )
#define OUTCRIT ( ASSERT(cri==1), cri++ )

void message(const char *)

extern CriticalSection (act)

struct Key
{
    DMOID userID;
    DMOID fromHash;

    BOOL operator==(const Key& k) const
    {
        return userID == k.userID && fromHash == k.fromHash;
    }

    void setID(user "u")
    {
        if (u.userID)
            userID = u.userID;
        else
            userID = u->p;
    }

    void setFrom(const char *from)
    {
        fromHash = hashw(from);
    }
};

UINT HashKey(Key key)
{
    return key.userID * key.fromHash;
    // default identity hash - works for most primitive values
    // return (UINT)(void)(DMOID)key) >> 4;
}

struct Value
{
    DMOID adSent;
    DMOID time;
};

class Memory
{
public:
    Memory() : sent(100)
    {
        sent.InitHashTable(SZ);
    }

    void remember(Key& k, DMOID adID)
    {
        DMOID lookupKey = k;
        private:
        void purge();
        CHAPKEY, Key, Value, Values sent;
        memory;
    }

    // info, etc

```

```
// todo: nonunique hashes
// DMOID hash(const char *from, User "u")
//
// char buf[10];
// sprintf(buf, "%s", u->getID());
// CString s = buf;
// s = from;
// return hashw(s);
//

void Memory::remember(Key& k, DMOID adID)
{
    static int count;
    if (++count > 1000 ) {
        count = 0;
        purge();
    }

    Value v;
    v.adSent = adID;
    v.time = 1;
    sent.SetAt(k, v);

    DMOID Memory::lookup(Key& k)
    {
        Value value;
        if (sent.Lookup(k, value) ) {
            return value.adSent;
        }
        return 0;
    }

    void Memory::purge()
    {
        const LIMIT = 1000 * 60 * 60 * 24; // too much?
        if (sent.GetCount() > SZ ) {
            message("remember map > SZ");
        }

        DMOID now = 1;
        POSITION p = sent.GetStartPosition();
        while (p) {
            Key k;
            Value v;
            sent.GetNextAssoc(p, k, v);
            if (now - v.time > LIMIT )
                sent.RemoveKey(k);
        }
    }

    void rememberSending "ad, User "u, const char *fromDoc)
    {
        Crit c(last);
        // INCRIT
        Key k;
        k.setID(u);
        k.setFrom(fromDoc);
        memory.remember(k, ad->id);
        // OUTCRIT

        DMOID queryAdSent User "u, const char *fromDoc)
        {
            Crit c(last);
            // INCRIT
            Key k;
            k.setID(u);
            k.setFrom(fromDoc);
            DMOID d = memory.lookup(k);
            // OUTCRIT
            return d;
        }
    }

```

HIGHLY  
CONFIDENTIAL  
DC 069507

```

// a truly random distribution is used for them rather than
// leftovers.
static int testCounter;
if (testCounter % 4 == 0) { // just try every 4 to save CPU
    // test ad avail;
    lowestSI = 1051;
    int i = start;
    while (1) {
        Ad ad = *Ada.GetAt(i);
        if (ad.type == Test && ad.si < lowestSI && ad.criteriaOkIdb, user, page) {
            lowestSI = ad.si;
            adlowestSI = ad;
            if (i == start)
                break;
        }
        if (lowestSI == 1050)
            return adlowestSI;
    }
    lowestSI = SIMAX;
    adlowestSI = defaultAd;
    // Check remnants first. This way, we don't
    // have to do ad matching for any targeted ads
    // with high SI's.
    int i = start;
    while (1) {
        Ad ad = *Ada.GetAt(i);
        if (ad.type == Normal && !ad.isTargeted() && ad.si < lowestSI && ad.spreadOk(page)) {
            lowestSI = ad.si;
            adlowestSI = ad;
            if (i == start)
                break;
        }
        i = (i + 1) % nAds();
    }
    // this is temp; eventual all placements will have book rates
    // you'll want to remove this to get better performance (no ad matching
    // if remnant has worst SI).
    static int counter;
    if (++counter & 1) {
        // for ads with no booking amount.
        // allow a targeted ad to run sometimes
        if (lowestSI == 1100)
            lowestSI++;
    }
    // for ads where we don't care about B impressions.
    // bias in favor of targeted
    if (lowestSI == 1100)
        lowestSI++;
    // todo later: if ads are sorted by si (lowest first),
    // you can quit matching as soon as you find
    // one. Could be a good optimization.
    // do targeted
    i = start;
    while (1) {
        Ad ad = *Ada.GetAt(i);
        if (ad.type == Normal && ad.isTargeted() &&
            ad.spreadOk(page) &&
            ad.matches(user, page) &&
            ad.isPosResOkIdb, user) {
            // found a good one
            lowestSI = ad.si;

```

```

        adlowestSI = ad;
    }
    i = (i + 1) % nAds();
    if (i == start)
        break;
}
if (lowestSI > 1400) {
    // do either a better ad or an 1st dev ad
    static int counter;
    if (++counter % 5 == 0) {
        // do an 1st dev ad
        i = start;
        while (1) {
            Ad ad = *Ada.GetAt(i);
            if (ad.type == 1stDev && ad.criteriaOkIdb, user, page) {
                // found a good one
                adlowestSI = ad;
                break;
            }
            i = (i + 1) % nAds();
        }
        if (i == start)
            break;
    }
    else {
        // do better
        lowestSI = SIMAX;
        i = start;
        while (1) {
            Ad ad = *Ada.GetAt(i);
            if (ad.type == Better &&
                ad.si < lowestSI &&
                ad.criteriaOkIdb, user, page) {
                // found a good one
                adlowestSI = ad;
                lowestSI = ad.si;
            }
            i = (i + 1) % nAds();
        }
        if (i == start)
            break;
    }
}
return adlowestSI;
}
}
}

```

DC 069504

HIGHLY  
CONFIDENTIAL

```

// request.cpp
#include <data.h>
#include <d/toolkit/sock.h>
#include <request.h>
#include <d/toolkit/utf_util.h>

if defined(CONSOLE)
#include <stream.h>
endif

if defined(LAR)
extern ostream coutlog;
void Impression();
endif

extern CString gratuitous;

Request::Request(
    Connection *c,
    Verb v,
    const char *request,
    const sockAddr_in from,
    c[_c], request[_request], v[_v])
{
    uaddrp = from.sin_addr.s_addr;

    int opdir = 0;

    bool Request::sendFile(const char *fileName, const char *insertStr)
    {
        if defined(LAR)
        coutlog << "and " << fileName << " " << int_ntol (ln_addr) uaddrp << "\n";
        endif

        const char insertChar = '-';
        bool isSpider = FALSE;

        CString hdr = "HTTP/1.0 200 OK\r\nContent-Type: ";
        if (strlen(fileName) > 0) {
            hdr += "application/java\r\nContent-Length: ";
        }
        else if (strlen(fileName) > 0) {
            hdr += "image/gif\r\nContent-Length: ";
        }
        else {
            hdr += "text/html\r\nContent-Length: ";
        }
        if defined(LAR)
        Impression();
        endif

        int gnt = 0;
        if (strlen(request) > 0) {
            gnt = 1;
        }
        if (strlen(request) > 0) {
            gnt = 2;
        }
        if (strlen(request) > 0) {
            gnt = 3;
        }
        if (gnt > 0)
        {
            isSpider = TRUE;
            spider = " ";
            if defined(CONSOLE)
            cout << "..... Robot " << gnt << " .....\n";
            endif
        }
    }

    const BUFSIZE = 13000;
    char buf[BUFSIZE + 20];
    CStringException ex;
}

```

DC 069505

HIGHLY

CONFIDENTIAL

```

if v == GET || v == POST {
    if (strlen(fileName) > 0) {
        if (ex_m_cause == CStringException::AccessDenied) {
            sendError("404 Not Found (Access Denied)");
        }
        else if (ex_m_cause == CStringException::SharingViolation) {
            sendError("404 Not Found (Sharing Violation)");
        }
        else {
            sendError("404 Not Found");
            return FALSE;
        }
    }
    n = (strlen(buf) + BUFSIZE);
    isSpider = FALSE;

    // MAXN
    n = getFileSize(fileName);
    if n > 0 {
        sendError("404 Not Found");
        return FALSE;
    }
    ASSERT(n > 0 && n < BUFSIZE);

    char *p = buf;
    if (insertStr) {
        while (1) {
            p = strchr(p, insertChar);
            if p == 0 {
                break;
            }
            int i = strlen(insertStr);
            memmove(p + 1, p + 1, strlen(p + 1));
            memcpy(p, insertStr, i);
            p += i;
            n += i;
        }
    }

    if (isSpider) {
        if (gratuitous.isEmpty()) {
            if defined(CONSOLE)
            cout << "gratuitous empty. (?)\n";
            endif
        }
        else {
            buf[n] = 0;
            char *p = strchr(buf, "/BODY");
            if p {
                for (int i = 0; i < 20; i++) {
                    strcpy(p, gratuitous);
                    p = gratuitous.GetLength();
                    strcpy(p, "/BODY<HTML>");
                    n = (p - buf) + 1;
                }
            }
        }
    }
    else {
        if defined(CONSOLE)
        cout << "/body?\n";
        endif
    }

    char temp[100];
    int len, temp, 10; // content length
    hdr = temp;
    hdr += "\r\n";

    CStringException ex;
    if (v == GET || v == POST) {
        CString(buf, n);
        return TRUE;
    }
}

```

```

// match.cpp
// Ad Matching!
#include "stdafx.h"
#include "object.h"
#include "d/cookie/db.h"
#include "d/cookie/dbutil.h"
extern Ad *defaultAd;
extern Ad *badKeyErrorAd;
extern int nextAd;

int mda();

// Returns TRUE if this location is in region.
// bool location::inContest (region)
{
    if (region.country != 0 && countryCode != region.country)
        return FALSE;
    if (region.areaCode != 0 && areaCode != region.areaCode)
        return FALSE;
    if (region.state.isEmpty() && strcmp(state, region.state) != 0)
        return FALSE;
    if (region.zipCode.isEmpty())
        return TRUE;
    // zip
    CString myZip = zipCode.Left(5); // strip zip-4 for now
    CString regZip = region.zipCode.Left(5);
    CString regZipEnd = region.zipEnd.Left(5);
    if (regZipEnd.IsEmpty())
        return regZip == myZip;
    return myZip > regZip && myZip <= regZipEnd;
}

bool Ad::isExposureOK(Database db, User *user)
{
    serifment = 0;
    if (frequency == 0 || tdb == 0)
        return TRUE;
    int n;
    bool found;
    if (user->getId() == 0) {
        TRACE("user id=0\n");
        return FALSE;
    }
    Cursor c(db);
    c.Bind(SQL_C_LONG, &n, sizeof(n));
    char sql[512] = "select exposures from exposure where ad_id=";
    addValue(sql, id, FALSE);
    strcat(sql, " and user_id=");
    addValue(sql, user->getId(), FALSE);
    c.execute();
    found = c.fetchment();
    if (found) {
        if (n > frequency)
            return FALSE;
        serifment = n + 1;
        char sql[1024];
    }
}

```

DC 069502

HIGHLY  
CONFIDENTIAL

```

// update exposure set exposures=exposures+1 where ad_id=";
addValue(sql, id, FALSE);
strcat(sql, " and user_id=");
addValue(sql, user->getId(), FALSE);
db.execute();
return TRUE;
}

char sql[1024];
// insert exposure values:
addValue(sql, id);
addValue(sql, user->getId(), FALSE);
strcat(sql, " ");
db.execute();

return TRUE;

// Note: any matching required for non-targeted ads can be placed here.
// since this function is called for both targeting and untargeted
// ads.
bool Ad::isExposureOK(SitePage *sitePage)
{
    // is start time met?
    if (started) {
        time_t now;
        if (time(&now) < starttime)
            return FALSE;
        return TRUE;
    }
    // impressions OK?
    if (unknown > maxImpressions && maxImpressions != 0)
        return FALSE;
    if (isSpreadingEvenly() && sl == 1120)
        return FALSE;
    if (targetSites.isEmpty()) {
        if (sitePage == 0)
            return FALSE;
        bool v;
        bool found = targetSites.Lookup(sitePage->siteID, v);
        if (includes(v)) {
            // if we have pages to target too, ok if site
            // doesn't match (check if page does next)
            if (found && targetPages.isEmpty())
                return FALSE;
            else if (found)
                return FALSE;
        }
        return TRUE;
    }
    return TRUE;
}

// Does user and site match this ad's criteria?
bool Ad::matches(User *user, SitePage *sitePage)
{
    if (targetPages.isEmpty()) {
        if (sitePage == 0)
            return FALSE;
        bool v;
        bool found = targetPages.Lookup(sitePage->id, v);
        if (includes(v)) {
            if (found)
                return FALSE;
            else if (found)
                // excluding this page
                return FALSE;
        }
    }
    // Operating system
    DWORD o = 1 << (int) user->os;
}

```

```

if( !o & oo) == 0 )
    return FALSE;

// Browser
o = 1 << (int) user->browser;
if( !o & browser) == 0 )
    return FALSE;

// DomainType
int userISP = 0;
int dt = (int) user->domainType;
if( dt > (int) dtISPOrder ) {
    userISP = dt - (int) dtISPOrder + 1;
    dt = 0;
}

// ISP
o = 1 << userISP;
if( !o & isp) == 0 )
    return FALSE;

else {
    o = 1 << dt;
    if( !o & domainType) == 0 )
        return FALSE;
}

// location
if( !location) == 0 ) { // if ISP, don't know location (yet)
    if( userISP )
        return FALSE;
}

BOOL ok = FALSE;
for( int i = 0; i < nLocations; i++ ) {
    if( user->location.in( locations[i] ) ) {
        ok = TRUE;
        break;
    }
}

if( !ok )
    return FALSE;

// hour of day / day of week
if( !hourOfDay || !dayOfWeek || !dayOfWeek == 0x7f ) {
    return FALSE;
}

// EST time relative
time_t now;
time_t localTime(now);
t = localTime(now);

else {
    t = user->location.userRelativeTime();
    if( t == 0 )
        return FALSE;
}

if( !hourOfDay & t < t-cm_hour) == 0 )
    return FALSE;
if( !dayOfWeek & t < t-cm_wday) == 0 )
    return FALSE;

// sales
if( !salesVolume || !0x7fffffff ) {
    if( !salesVolume || !0x7fffffff )
        o = 1 << user->salesVolume;
    if( !o & salesVolume) == 0 )
        return FALSE;
}

// 8 employees
if( !employees || !0x7fffffff ) {
    o = 1 << user->employees;
    if( !o & employees) == 0 )
        return FALSE;
}

```

DC 069503

HIGHLY  
CONFIDENTIAL

```

// SIC
if( !sicCodes ) {
    BOOL ok = FALSE;
    int i = 0;
    while( i ) {
        if( i >= nsicCodes ) {
            // no match
            return FALSE;
        }
        sicCodes.pattern = sicCodes[i];
        user->sicCodes.reset();
        sicCode sc;
        while( user->sicCodes.getNext(sc) ) {
            if( pattern.matches(sc) ) {
                break;
            }
        }
        if( !ok )
            break;
    }
}

// Site and page categories
// Do last, because this is expensive (disk hit)
if( !siteCategories.isEmpty() ) {
    BOOL vi;
    if( !sitePage == 0 )
        return FALSE;
    sitePage->loadCategories();
    for( int i = 0; i < sitePage->categories.GetSize(); i++ )
        if( !siteCategories.Lookup(sitePage->categories.data[i], vi) )
            return TRUE;
    return FALSE;
}

return TRUE;

}

inline BOOL Ad::criteriaOK(Database db, User *user, SitePage *page)
{
    return spreadOK(page) &&
        (!isTargeted() ||
         matches(user, page) && exposureOK(db, user))
    ;
}

// todo: if reload ads, need to handle the fact that
// one may still be in use and can't just delete.
// (crit sect released during sending of file.)
// Ad::Ad::getAd(Database db, User *user, SitePage *page, BOOL increment)
{
    const SIMAX = 1000000;
    if( user->uniqueness < unlikely )
        return defaultAd;
    if( page == 0 ) {
        if( !badKeyErrorAd )
            return badKeyErrorAd;
        ASSERT(FALSE);
    }
    if( !increment )
        nextAd = (nextAd + 1) % nAd();
    int lowestSi;
    Ad *adLowestSi;
    const int asize = nextAd;
    // Do a test ad, if appropriate. Always do these first so that

```

```
sendit  
    errlog.Flush();  
}  
// temp: just return first ad (ISS)  
//return new Ad( ada.ElementAt(0) );  
    return new Ad( "defaultAd" );  
// return 0;  
}  
sendit  
sendit  
sendit
```

DC 069500

HIGHLY  
CONFIDENTIAL

```
// cookie.cpp
//
#include "acdata.h"
#include "object.h"
//.....
// Cookie
const Cookie::operator(const char *)
{
    sscanf("%lx", &value);
    return "chle";
}

//static/
Cookie Cookie::allloc(idword userid)
{
    ASSERT(userid != 0);
    Cookie k;
    k.value = userid;
    return k;
}

// Get value for a particular cookie name from the HTTP header
// hdr - points to the Cookie field in the header
void Cookie::getfromheader(const char *hdr, const char *name)
{
    hdr += 7; // skip "Cookie:"
    const char *p = strchr(hdr, '\r');
    if (p) {
        Ctering nm = name;
        nm += "...";
        const char *q = strchr(hdr, nm);
        if (q && q < p)
            *this = q + nm.GetLength();
    }
}
```

DC 069501  
HIGHLY  
CONFIDENTIAL

```

//domainName
text << "<b>location: </b>"
if( location.country == 356 ) {
    text << "US";
} else {
    text << "country # " << location.country;
}
text << "\n\n";

text << "<b>job function:</b>" << "\n\n";
text << "<b>gender: </b>"
if( gender == "m" || gender == "n" )
    text << "Male";
else if( gender == "f" || gender == "g" )
    text << "Female";
else
    text << "?";
text << "\n\n";
text << "<b>?";

Domain d = Domain.lookupDomain(ip);
if( d == 0 ) {
    text << "no company information available.</b>";
} else {
    text << "<b>domain name: </b>" << (const char *) d->domain << "\n\n";
    text << "<b>bu. name: </b>" << (const char *) d->name << "\n\n";
    text << "<b>address: </b>" << (const char *) d->address[0] << "\n\n";
    for( int i = 1; i < MAXDRI; i++ ) {
        if( i < address[i] - isEmpty() ) {
            text << " " << (const char *) d->address[i] << "\n\n";
        }
    }
    text << "<b>contact: </b>" << (const char *) d->contact[0] << "\n\n";
    for( i = 1; i < NCONTACT; i++ ) {
        if( i < contact[i] - isEmpty() ) {
            text << " " << (const char *) d->contact[i] << "\n\n";
        }
    }
    text << "<b>industries: </b>\n\n";
    {
        sICodes.reset();
        sICode sCI;
        while( sICodes.getNext(ac) ) {
            text << " " << ac.asTextFullyPadded() << "\n\n";
        }
    }
    for( i = 0; i < MAXSICS; i++ )
        if( d->sICodes[i] )
            text << d->sICodes[i] << " ";
    text << "\n\n";
    text << "<b>no. empl.: </b>"
    if( nEmployees )
        text << nEmployees;
    else
        text << "less than 25 (unknown)";
    text << "\n\n";
    text << "<b>volume: </b>"
    if( salesVolume )
        text << salesVolume;
    else
        text << "less than 5MM (unknown)";
    delete d;
}

text << "<b>?";
text << "<b>you are interested in the following:</b>\n\n";
text << "<b>Interest Level Category: Description</b>";
text << ".....</b>";
}

//DMDRD level;
//String category;

```

DC 069498

**HIGHLY  
CONFIDENTIAL**

```

        Casting desc;
        Cursor c(dbi);
        c.bind(SOL_C_LONG, slevel, 4);
        c.bind(category);
        c.bind(desc);
        char sql[512];
        sprintf(sql,
            "select interest_level, category_name from interests_user_interests\
            where interests_id=interest_id and user_id=11d\
            order by interest_level DESC, userID);");
        c.execute(sql);
        while (c.fetchNext()) {
            char buf[512];
            sprintf(buf, "%ld ", level);
            text += buf;
            text += category + " - " + desc + "\n\n";
        }
        db.commit();
    }

void User::getNetworkInfo(Database db, bool *timedOut)
{
    if (!ip == 0 ) {
        ASSERT(FALSE);
        return;
    }

    // if domainType != dUnknown ) {
    //     // got it from header info
    //     // if ISP/OSP, location and sales, etc. don't apply.
    //     // if we have done a tracerf, location does apply.
    //     // for ISP/OSP's.
    //     if (domainType != dNetcom ) // did tracerf for netcom
    //         return;
    // }

    // Note: do the following for all domain types to at least get country.
    NetworkNumber n;
    n = justNetworkNumber(ip);

    char buf[1024] =
        "select domain_type,sales,num_employees,nic,country,state,zipcode,areacode from networks\
        cursor c(dbi) {"
        if (domainType < dAOL ) {
            c.bind(SOL_C_LONG, idomainType, sizeof(domainType));
            c.bind(SOL_C_LONG, salesVolume, sizeof(salesVolume));
            c.bind(SOL_C_LONG, numEmployees, sizeof(numEmployees));
            c.bind(SOL_C_LONG, nicCode, sizeof(nicCode));
        } else {
            strcpy(buf, "select country,state,zipcode,areacode from networks where netnumber=");
            sprintf(buf, "%qstr()", n);
        }
        c.bind(SOL_C_LONG, allocation.country, sizeof(allocation.country));
        c.bind(location.state);
        c.bind(location.zipCode);
        c.bind(SOL_C_LONG, allocation.areaCode, sizeof(allocation.areaCode));
        if (timedout != 0 )
            c.setTimeOut(1);
        c.execute(buf);
        if (c.getTimeOut() )
            *timedout = TRUE;
        else
            c.fetchNext();
    }

    if ( uniqueness == unknown || (int) domainType > (int) dAOL )
        uniqueness = unlikely;

    if (domainType >= dAOL ) {
        allocation = 0;
        numEmployees = 0;
        nicCode = makeNull();
        if (domainType != dNetcom && domainType != dISOther ) {

```

```

// don't know location, except country
location.state.Empty();
location.zipCode.Empty();
location.areaCode = 0;

}
else {
    statusCode.checkNull();
}

if (defined_DERIVE)
    const char cCookie[] = "Cookie:";

void User::initWithConst char *verstr)
{
    int v1 = 0, v2 = 0;
    secantVerstr;
    *v1, *v2, *v3, *v4;
    bVer1 = v1;
    bVer2 = v2;
}

// Use "u" to lookup user by ID (DND user ID)
User *u = new User;
return u;

User *User::lookupUserByAddress(DND ip)
{
    DND ipDND = networkNodeTable.getUserID(ip, FALSE);
    if (ipDND == 0) {
        // Try to get domain info at least. Note: if user is uniquely
        // identifiable, derive data process will create a record for the
        // user as soon as it gets a chance.
        userID = networkNodeTable.getUserID(networkNodeNumber(ip), TRUE);
        if (userID) {
            return lookupUserByID(userID);
        }
        return 0;
    }
}

extern defaultNode;

User *User::lookupUser(Database db, DND ip, const char *requestHdr, BOOL loadDemographics,
    BOOL _timedOut = 0, BOOL _realTime = 0, BOOL _timedOut = 0)
{
    // .....
    // get cookie for lookup
    cookie cookie;

    const char *ch = strstr(requestHdr, cCookie);
    if (ch)
        cookie.getFromHeader(ch, "IP");

    // .....
    // lookup
    User *u = 0;

    if (cookie.isNull()) {
        if (_timedOut) {
            u = new User;
            u->uniqueness = uVer;
            u->ip = ip;
            u->userID = cookie.value;
            u->timedOut = TRUE;
        }
    }
}

```

DC 069499  
HIGHLY  
CONFIDENTIAL

```

}
else {
    // lookup by cookie
    u = lookupUserByID(db, cookie.value, timeout);
    if (u) {
        u->uniqueness = uVer;
        u->ip = ip;
    }
    else {
        if (defaultNode) {
            // db conn down
            u = new User;
            u->uniqueness = uVer;
            u->ip = ip;
            u->userID = cookie.value;
        }
        // Couldn't find user record, we will need to
        // assign a new cookie. Do not load by IP, because
        // we don't want this user sharing a record
        // with others without cookies.
        // Note: generally, this shouldn't happen.
        cookie.value = 0;
    }
}

else if (!timedOut) {
    u = lookupUserByAddress(db, ip, timeout);
    if (u) {
        u->ip = ip;
        u->hasCookie = FALSE;
    }
}

if (u == 0) {
    // make a default user object
    u = new User;
    // u->uniqueness = uVer;
    u->ip = ip;
    u->timedOut = _timedOut;

    u->headerDerive(requestHdr);
    if (cookie.isNull())
        u->hasCookie = TRUE;

    if (loadDemographics && !_timedOut)
        u->getNetworkInfo(db, realTime ? u->timedOut : 0);

    return u;
}

// .....
// SitePage
Ad *Ad::findSentToUser *user, const char *fromDoc)
{
    DND adNum = queryAdSent(user, fromDoc);
    for (int i = 0; i < nAdes(); i++) {
        Ad *ad = ads.GetAt(i);
        if (ad->id == adNum)
            return new Ad(ad);
    }

    if (badKeyErrorAd && adNum == badKeyErrorAd->id)
        return badKeyErrorAd;

    if (user->uniqueness == unlikely) {
        if (definedErrorLog)
            errorLog << "findSentTo failed uniqueness=unlikely\n";
            errorLog << "user = " << user->userID << "\n";
            errorLog << "from doc = " << fromDoc << "\n";
    }
}

```